Universidad del Valle de Guatemala
Introducción a la Organización de Computadoras
Ing. Marta Ligia Naranjo de Ortiz
Sección: 10

Proyecto Final

-Investigación Corta-

Integrantes:	Carné:
Jonathan A. de los Santos Chonay	09375
Cristian Alberto Peláez Pérez	09376
Donald Antonio Velázquez Aguilar	09379

Investigación Corta

Variables y objetos utilizados

Para llegar a la solución se hizo un diseño de la máquina Intel Simplificada, este diseño consistió en definir el tamaño de memoria, definir los registros visibles y no visibles y el tamaño de palabra, que comprende las características principales de la máquina virtual y en base a las cuales se diseña el traductor y el compilador. La memoria se diseñó de forma que se tienen 2^16 localidades de memoria, cada una con una capacidad de 16 bits. Los registros utilizados son 8 nombrados: AX, BX, CX, DX, EX, FX, GX, HX. También se muestra el IP (Instruction Pointer) que es el equivalente al PC en la LC-3.

Para representar la memoria se utilizan objetos de la clase Memoria, esta clase tiene los atributos nombre y contenido, los objetos de ésta clase se agregaron a un ArrayList llamado memoria. Los registros son análogos a la memoria ya que existe una clase de tipo Registro, los cuales poseen un nombre y un contenido como atributos, y luego se agregan a un ArrayList llamado registros. Las clases Memoria y Registro poseen un método toString() que es utilizado por el constructor del JList para mostrarlos en la interfaz gráfica.

Se tiene una clase Traductor, cuya función es interpretar cada línea del archivo con código ensamblador (archivo con extensión .asm) y "traducir" las instrucciones de ésta a lenguaje de máquina, con ayuda de ésta clase se crea el archivo con extensión .bin. que contiene todas esas instrucciones. La clase Compilador se utiliza para interpretar cada una de las instrucciones almacenadas en memoria. Esta clase recibe en su constructor una línea en lenguaje binario, la cual es cargada desde las localidades de memoria.

En la clase principal del programa IMaquina se tienen los métodos para ir actualizando los JList que contienen los registros y las localidades de memoria. También maneja los métodos para imprimir mensajes en consola, cambios de valores de memoria y registros de forma manual, correr el programa de forma continua y paso a paso, detener el programa, cargar un programa fuente, reiniciar la máquina virtual, saltar a una localidad de memoria especificada y mostrar información acerca de la máquina virtual.

Para permitir al programa correr o detenerse se utiliza una variable dentro de la clase IMaquina llamada HLT (no confundir con la instrucción HLT). Cuando esta variable tiene un valor lógico false, el programa corre normalmente hasta detenerse por la instrucción HLT, en la cual se cambia el valor de esta variable y se detiene el programa. La opción para detener el programa consiste en un cambio del valor lógico de la variable HLT a true, con lo cual se detiene el programa.

Descripción del curso lógico de la solución

La forma en como se llega a la solución del problema es el siguiente: primero se debe cargar un programa escrito en el lenguaje ensamblador de la Intel Simplificada (archivo con extensión .asm), éste es traducido a un archivo binario (instrucciones en lenguaje de máquina en un archivo con extensión .bin) y a un archivo object (archivo con extensión .obj). Luego las instrucciones son cargadas a las localidades de memoria, iniciando en la dirección indicada con la instrucción ORG de forma similar a la LC-3. Todo esto se hace durante el proceso de cargar un archivo. Cuando se ejecuta la máquina virtual, el IP contiene la dirección 3000H (la H se refiere a un valor Hexadecimal, así como el caracter B al final de un valor numérico indica que se trata de un valor binario), cuando se carga un archivo, el valor del IP cambia automáticamente a la dirección de memoria donde inician el programa.

Despues de cargar las instrucciones a la memoria, y crear los archivos .bin y .obj, la máquina espera a que se le indique el iniciar a ejecutar las instrucciones, la ejecución de las instrucciones puede ser de forma continua o por pasos.

La ejecuciones de las instrucciones cargadas en la memoria sigue el mismo proceso que las instrucciones en la LC-3. Primero se obtiene la dirección a la que apunta el IP y luego se aumenta en uno el IP, con esta dirección obtenida del PC se extrae el String contenido de la localidad de memoria a la que apunta el PC, este String es el código en lenguaje de máquina. El código extraído de cada localidad de memoria es interpretado por la clase Compilador, en ésta clase se hacen los cambios de los registros, y de las localidades de memoria, dependiendo de la instrucción que se esté analizando.

Métodos principales implementados

cargarPrograma( ): este método se encarga de desplegar un JFileChooser, con el que se selecciona el programa en lenguaje ensamblador. Este método se encarga tambien de llamar a la clase Traductor, con lo que se traduce el contenido del lenguaje ensamblador a lenguaje binario y se crea el archivo binario, aquí también se llama al método guardar( ) que genera el archivo obj.

iniciarRegistros( ): con este método se inician los valores de registros, y se añaden al ArrayList que los contiene, durante la construcción de los registros se les asigna un nombre único, que se utiliza para identificar cada registro, y el valor de contenido es cero.

IniciarMemoria( ): es el método encargado de iniciar la memoria durante la ejecución de la máquina virtual. Este método asigna a cada memoria un valor de localidad único, con el que se identifica a cada localidad de memoria, y un contenido, que al inicio corresponde a la cadena “0000000000000000”.

reiniciarMemoria( ): cambia el contenido de todas las localidades de memoria por la cadena “0000000000000000”. Este método es utilizado para reiniciar la máquina virtual.

printConsola(String mensaje): este método “imprime” en consola el String que entra como parámetro. Este método es implementado con ayuda del método setText( ) de la clase JTextArea. En éste se conserva los mensajes anteriores de la consola, por lo que no puede se utilizado para limpiar o reiniciar la consola.

limpiarConsola( ): este método cambia el contenido de la consola, por una cadena de caracteres vacía, es decir, deja la consola con ningún caracter.

reiniciar( ): este método se encarga de cambiar los valores de los registros, memoria, y consola de la máquina virtual a los valores iniciales que ésta tiene cuando se inicia. Este método no es más que un llamado a los métodos reiniciarMemoria( ), iniciarRegistros( ) y limpiarConsola( ).

cambiarContenido(int localidad, String contenido), cambiarContenido(String nombre, int contenido): estos métodos están presentes en las clases: la clase PMemoria y la clase PRegistro respectivamente. La función de éstos es cambiar el valor que contiene la memoria o registro, por el parámetro que entra en el método. El primer parámetro de entrada es un int para la memoria y un String para el registro, y se utiliza para identificar la localidad. El segundo parámetro de entrada es un int para los registros y es un String para las localidades de memoria, que son los valores por los que se cambiará el contenido de cada objeto. Estos métodos requieren que se indique el nuevo valor de entrada, por lo que se requiere de la existencia de JDialog para el ingreso de los valores de forma manual. La importancia de éste método radica en que es utilizado para cambiar los valores de los registros y de la memoria por el programa cargado en la memoria durante su ejecución.

darMemoria(int localidad): este método permite acceder a una localidad de memoria, y obtener el contenido de la misma. La localidad a la que se desea acceder está definida por el int que entra como parámetro en el método. El valor retornado por éste método es un String el cual es una instrucción en binario del método.

darRegistro(String nombre): la finalidad de éste método es similar a la del método anterior, pero su acceso es a los registro. El acceso a los registros se hace a través del String que entra como parámetro, el cual se utiliza para compararlo con los nombres de cada registro.

porPaso( ): este método está implementado en la clase IMaquina, su función es ejecutar una instrucción del programa cargado, es decir, el programa accede a la instrucción almacenada en la localidad de memoria que apunta el IP, luego el IP se aumenta en uno, y la cadena de instrucción obtenida es analizada por el Compilador.

correr( ): este método, también en la clase IMaquina, tiene una función similar a la del método antes descrito, pero la diferencia radica en que este ejecutará todas las instrucciones una tras otra, hasta encontrar una instrucción HLT, la cual detiene el proceso.

detener( ): este método se encarga de detener el proceso de la máquina virtual cuando está ejecutando las instrucciones del programa cargado en memoria. Dicho de otra forma, este programa "detiene" al proceso correr( ).

salto(int localidad): este método es el encargado de cambiar el index (elemento seleccionado) del JList que contiene la Memoria. Este método requiere de un int como entrada, el cual es utilizado para indexar el elemento seleccionado en el JList.

hexADec(char[ ] hexadecimal): este método se encarga de convertir un arreglo de caracteres considerada como hexadecimal (caracteres de 0 a 9 y de A a F) en un valor decimal. Este método es utilizado repetidas veces en el código y cuenta con modificaciones en algunas clases.

binADec(char[ ] binario): este método convierte un arreglo de caracteres binarios (caracteres 0 y 1) en un valor decimal. Este método, al igual que el anterior, es bastante utilizado por las transformaciones de binario a decimal que se tiene que hacer durante la interpretación del código.

sext(String valor): este método se encarga de hacer extensión de signo de un valor binario o hexadecimal para que sean 16 caracteres o 4 caracteres respectivamente. Además analiza si la cadena contiene menos de los caracteres máximos permitidos en cada tipo de String (hexadecimal y binario), de ser así, toma únicamente los últimos caracteres en cuenta, y descarta los primeros caracteres que están en exceso.

Conjunto de Instrucciones

La siguiente tabla presenta el conjunto de instrucciones implementadas en la Maquina virtual Intel Simplificada. La descripción de cada instrucción se hace más detallada en el manual de usuario.

Tabla 1. Conjunto de instrucciones implementadas en la Intel

INST.	OPCODE				11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	0	DST			SRC1			0	0	0	SRC2		
ADD	0	0	0	0	DST			SRC			1	Inmediato				
NOT	0	0	0	1	DST			SRC			1	1	1	1	1	1
AND	0	0	1	0	DST			SRC1			0	0	0	SRC2		
AND	0	0	1	0	DST			SRC			1	Inmediato				
NEG	0	0	1	1	DST			SRC			1	1	1	1	1	1
OR	0	1	0	0	DST			SRC1			0	0	0	SRC2		
OR	0	1	0	0	DST			SRC			1	Inmediato				
XOR	0	1	0	1	DST			SRC1			0	0	0	SRC2		
XOR	0	1	0	1	DST			SRC			1	Inmediato				
JMP	0	1	1	0	0	0	0	SRC			0	0	0	0	0	0
HLT	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IN	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
MOV	1	0	0	1	DST			SRC			1	1	1	1	1	1
J	1	0	1	0	n	z	p	PCOffset9								
MLT	1	0	1	1	DST			SRC1			0	0	0	SRC2		
MLT	1	0	1	1	DST			SRC			1	Inmediato				
LEA	1	1	0	0	DST			PCOffset9								
LDS	1	1	0	1	SRC			PCOffset9								

Conclusiones

- Durante la realización de éste proyecto se mejoró el trabajo en equipo, y se logró obtener un mayor conocimiento respecto al funcionamiento de una Máquina Virtual Intel Simplificada.

- Con la realización de éste proyecto se pudo aplicar los conocimientos adquiridos durante el desarrollo de éste y otros cursos.

- Se mejoró el aprendizaje obtenido debido a la integración de conocimientos de diferentes cursos y al aporte de ideas por parte de los integrantes del grupo.

- Se desarrolló una Máquina Virtual Intel Simplificada, capaz de ejecutar un conjunto de instrucciones programadas desde un editor de texto simple.

Bibliografía

1. Yale N. Patt, Sanjay J. Patel. Tr: Lorena Peralta R. 2005. Introducción a los Sistemas de Cómputo. Segunda edición. McGraw-Hill, México. 632 pp.

2. Peter Abel. 2001. IBM PC Assembly Language and Programming. Prentice Hall. Quinta edición. United States of América. 545 pp.

3. Zack Smith. 2009. The Intel 8086/8088/80186/80286/80386/80486 Instruction Set. Consultado en: <Home.comcast.net/ fbui/intel.html>